NAG Library Function Document

nag_dgttrs (f07cec)

1  Purpose

nag_dgttrs (f07cec) computes the solution to a real system of linear equations $AX = B$ or $A^TX = B$, where $A$ is an $n$ by $n$ tridiagonal matrix and $X$ and $B$ are $n$ by $r$ matrices, using the $LU$ factorization returned by nag_dgttrf (f07cdc).

2  Specification

#include <nag.h>
#include <nagf07.h>

void nag_dgttrs (Nag_OrderType order, Nag_TransType trans, Integer n, Integer nrhs, const double dl[], const double d[], const double du[],
                   const double du2[], const Integer ipiv[], double b[], Integer pdb,
                   NagError *fail)

3  Description

nag_dgttrs (f07cec) should be preceded by a call to nag_dgttrf (f07cdc), which uses Gaussian elimination with partial pivoting and row interchanges to factorize the matrix $A$ as

$$A = PLU,$$

where $P$ is a permutation matrix, $L$ is unit lower triangular with at most one nonzero subdiagonal element in each column, and $U$ is an upper triangular band matrix, with two superdiagonals. nag_dgttrs (f07cec) then utilizes the factorization to solve the required equations.

4  References


5  Arguments

1:  order – Nag_OrderType
    
    On entry: the order argument specifies the two-dimensional storage scheme being used, i.e., row-
    major ordering or column-major ordering. C language defined storage is specified by
    order = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed
    explanation of the use of this argument.

    Constraint: order = Nag_RowMajor or Nag_ColMajor.

2:  trans – Nag_TransType
    
    On entry: specifies the equations to be solved as follows:

    trans = Nag_NoTrans
    Solve $AX = B$ for $X$.

    trans = Nag_Trans or Nag_ConjTrans
    Solve $A^TX = B$ for $X$.

    Constraint: trans = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.
On entry: $n$, the order of the matrix $A$.

**Constraint:** $n \geq 0$.

4:  
$nrhs$ – integer

On entry: $r$, the number of right-hand sides, i.e., the number of columns of the matrix $B$.

**Constraint:** $nrhs \geq 0$.

5:  
$dl[dim]$ – const double

**Note:** the dimension, $dim$, of the array $dl$ must be at least $\max(1, n-1)$.

On entry: must contain the $(n-1)$ multipliers that define the matrix $L$ of the $LU$ factorization of $A$.

6:  
$d[dim]$ – const double

**Note:** the dimension, $dim$, of the array $d$ must be at least $\max(1, n)$.

On entry: must contain the $n$ diagonal elements of the upper triangular matrix $U$ from the $LU$ factorization of $A$.

7:  
$du[dim]$ – const double

**Note:** the dimension, $dim$, of the array $du$ must be at least $\max(1, n-1)$.

On entry: must contain the $(n-1)$ elements of the first superdiagonal of $U$.

8:  
$du2[dim]$ – const double

**Note:** the dimension, $dim$, of the array $du2$ must be at least $\max(1, n-2)$.

On entry: must contain the $(n-2)$ elements of the second superdiagonal of $U$.

9:  
$ipiv[dim]$ – const integer

**Note:** the dimension, $dim$, of the array $ipiv$ must be at least $\max(1, n)$.

On entry: must contain the $n$ pivot indices that define the permutation matrix $P$. At the $i$th step, row $i$ of the matrix was interchanged with row $ipiv[i-1]$, and $ipiv[i-1]$ must always be either $i$ or $(i+1)$, $ipiv[i-1] = i$ indicating that a row interchange was not performed.

10:  
$b[dim]$ – double

**Note:** the dimension, $dim$, of the array $b$ must be at least

$\max(1, pdb \times nrhs)$ when $order =$ Nag_ColMajor;

$\max(1, n \times pdb)$ when $order =$ Nag_RowMajor.

The $(i,j)$th element of the matrix $B$ is stored in

$b[(j-1) \times pdb + i - 1]$ when $order =$ Nag_ColMajor;

$b[(i-1) \times pdb + j - 1]$ when $order =$ Nag_RowMajor.

On entry: the $n$ by $r$ matrix of right-hand sides $B$.

On exit: the $n$ by $r$ solution matrix $X$.

11:  
$pdB$ – integer

On entry: the stride separating row or column elements (depending on the value of $order$) in the array $b$. 

**Mark 25**
Constraints:

if order = Nag_ColMajor, pdb ≥ max(1, n);
if order = Nag_RowMajor, pdb ≥ max(1, nrhs).

12: fail – NagError

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM
On entry, argument ⟨value⟩ had an illegal value.

NE_INT
On entry, n = ⟨value⟩.
Constraint: n ≥ 0.

On entry, nrhs = ⟨value⟩.
Constraint: nrhs ≥ 0.

On entry, pdb = ⟨value⟩.
Constraint: pdb > 0.

NE_INT_2
On entry, pdb = ⟨value⟩ and n = ⟨value⟩.
Constraint: pdb ≥ max(1, n).

On entry, pdb = ⟨value⟩ and nrhs = ⟨value⟩.
Constraint: pdb ≥ max(1, nrhs).

NE_INTERNAL_ERROR
An internal error has occurred in this function. Check the function call and any array sizes. If the
call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE
Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

7 Accuracy

The computed solution for a single right-hand side, x̂, satisfies an equation of the form

\[(A + E)\hat{x} = b,\]

where

\[\|E\|_1 = O(\epsilon)\|A\|_1\]

and \(\epsilon\) is the machine precision. An approximate error bound for the computed solution is given by
where \( \kappa(A) = \|A^{-1}\|_1 \|A\|_1 \), the condition number of \( A \) with respect to the solution of the linear equations. See Section 4.4 of Anderson et al. (1999) for further details.

Following the use of this function nag_dgtcon (f07cgc) can be used to estimate the condition number of \( A \) and nag_dgtrfs (f07chc) can be used to obtain approximate error bounds.

8 Parallelism and Performance

Not applicable.

9 Further Comments

The total number of floating-point operations required to solve the equations \( AX = B \) or \( A^T X = B \) is proportional to \( nr \).

The complex analogue of this function is nag_zgtrfs (f07csc).

10 Example

This example solves the equations

\[
AX = B,
\]

where \( A \) is the tridiagonal matrix

\[
A = \begin{pmatrix}
3.0 & 2.1 & 0 & 0 & 0 \\
3.4 & 2.3 & -1.0 & 0 & 0 \\
0 & 3.6 & -5.0 & 1.9 & 0 \\
0 & 0 & 7.0 & -0.9 & 8.0 \\
0 & 0 & 0 & -6.0 & 7.1
\end{pmatrix}
\]

and

\[
B = \begin{pmatrix}
2.7 & 6.6 \\
-0.5 & 10.8 \\
2.6 & -3.2 \\
0.6 & -11.2 \\
2.7 & 19.1
\end{pmatrix}.
\]

10.1 Program Text

/* nag_dgttrs (f07cec) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
* */
#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf07.h>

int main(void)
{
  int exit_status = 0, i, j, n, nrhs, pdb;
  double *b = 0, *d = 0, *dl = 0, *du = 0, *du2 = 0;
  Integer *ipiv = 0;
  NagError fail;
  Nag_OrderType order;

  #ifdef NAG_COLUMN_MAJOR
  #define B(I, J) b[((J-1)*pdb + I - 1]
  order = Nag_ColMajor;
  #endif

  // ...


#ifdef B(I, J) b[(I-1)*pdb + J - 1]

#else

#define B(I, J) b[(I-1)*pdb + J - 1]

#endif

order = Nag_RowMajor;

#endif

INIT_FAIL(fail);

printf("nag_dgttrs (f07cec) Example Program Results\n\n");

/* Skip heading in data file */

#ifdef _WIN32

scanf_s("%*[\n");
#else

scanf("%*[\n");
#endif

#ifdef _WIN32

scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[\n"); &n, &nrhs);
#else

scanf("%"NAG_IFMT"%"NAG_IFMT"%*[\n"); &n, &nrhs);
#endif

if (n < 0 || nrhs < 0)
{

printf("Invalid n or nrhs\n");
exit_status = 1;
goto END;
}

/* Allocate memory */

if (!b = NAG_ALLOC(n * nrhs, double)) ||
!d = NAG_ALLOC(n, double)) ||
!(dl = NAG_ALLOC(n-1, double)) ||
!(du = NAG_ALLOC(n-1, double)) ||
!(du2 = NAG_ALLOC(n-2, double)) ||
!(ipiv = NAG_ALLOC(n, Integer)))
{

printf("Allocation failure\n");
exit_status = -1;
goto END;
}

#ifdef NAG_COLUMN_MAJOR

pdb = n;
#else

pdb = nrhs;
#endif

/* Read the tridiagonal matrix A from data file */

#ifdef _WIN32

for (i = 0; i < n - 1; ++i) scanf_s("%lf", &du[i]);
#else

for (i = 0; i < n - 1; ++i) scanf("%lf", &du[i]);
#endif

#ifdef _WIN32

scanf_s("%*[\n");
#else

scanf("%*[\n");
#endif

#ifdef _WIN32

for (i = 0; i < n; ++i) scanf_s("%lf", &d[i]);
#else

for (i = 0; i < n; ++i) scanf("%lf", &d[i]);
#endif

#ifdef _WIN32

scanf_s("%*[\n");
#else

scanf("%*[\n");
#endif

#ifdef _WIN32

for (i = 0; i < n - 1; ++i) scanf_s("%lf", &dl[i]);
#else

for (i = 0; i < n - 1; ++i) scanf("%lf", &dl[i]);
#endif

Mark 25

f07 – Linear Equations (LAPACK)

f07cec

f07cec.5
/* Read the right hand matrix B */
for (i = 1; i <= n; ++i)
#ifdef _WIN32
    for (j = 1; j <= nrhs; ++j) scanf_s("%lf", &B(i, j));
#else
    for (j = 1; j <= nrhs; ++j) scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
/* Factorize the tridiagonal matrix A using nag_dgttrf (f07cdc). */
nag_dgttrf(n, dl, d, du, du2, ipiv, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dgttrf (f07cdc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Solve the equations AX = B using nag_dgttrs (f07cec). */
nag_dgttrs(order, Nag_NoTrans, n, nrhs, dl, d, du, du2, ipiv, b, pdb, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dgttrs (f07cec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print the solution using nag_gen_real_mat_print (x04cac). */
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, b, pdb, "Solution(s)", 0, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
END:
NAG_FREE(b);
NAG_FREE(d);
NAG_FREE(dl);
NAG_FREE(du);
NAG_FREE(du2);
NAG_FREE(ipiv);
return exit_status;
#endif
10.2 Program Data

nag_dgttrs (f07cec) Example Program Data

\[
\begin{array}{cccc}
5 & 2 & : & n \text{ and } \text{nrhs} \\
2.1 & -1.0 & 1.9 & 8.0 \\
3.0 & 2.3 & -5.0 & -0.9 & 7.1 \\
3.4 & 3.6 & 7.0 & -6.0 & : \text{ matrix A} \\
2.7 & 6.6 \\
-0.5 & 10.8 \\
2.6 & -3.2 \\
0.6 & -11.2 \\
2.7 & 19.1 & : \text{ matrix B}
\end{array}
\]

10.3 Program Results

nag_dgttrs (f07cec) Example Program Results

Solution(s)

\[
\begin{array}{cc}
1 & 2 \\
1 & -4.0000 & 5.0000 \\
2 & 7.0000 & -4.0000 \\
3 & 3.0000 & -3.0000 \\
4 & -4.0000 & -2.0000 \\
5 & -3.0000 & 1.0000
\end{array}
\]